Preliminary Study toward Intelligent Run-time Resource Management Techniques for Large Multi-Core Architectures

Dong-In Kang, Jinwoo Suh, Janice O. McMahon, and Stephen P. Crago
University of Southern California – Information Sciences Institute
{dkang, jsuh, jmcmahon, crago}@isi.edu

Introduction

The rising processor density and the advent of multi-core architectures have increased the amount of on-chip processing resources at a rate commensurate with Moore's Law. As the number of resources on-chip increases, it becomes more likely that multiple applications will have to share those resources, applications whose resource requirements are independent of each other and vary in response to their own environmental stimuli [1]. Maintaining high levels of performance on these applications will require efficient, run-time arbitration of on-chip resources. Such arbitration will require the ability to allocate and de-allocate dynamically resources such as cores and network links, and will defy the static mappings and stove-piped parallelism of previous architectures.

This abstract motivates the need for intelligent run-time resource management techniques for large multi-core architectures. The need for these techniques arises from two key factors: resource fragmentation and application mapping. Resource fragmentation occurs when an application is mapped to an irregular and spatially discontinuous region of cores. Application mapping refers to the assignment of tasks to cores in the processor such that high performance is achieved by minimizing communication latencies between tasks in the application. To motivate the need for intelligent run-time systems, we present a model of an actual application and specify how its resource requirement might vary over time. We show how resource fragmentation can occur under those specified variations in resource requirements. Using a high-level simulator developed at USC/ISI, we then show how resource fragmentation and sub-optimal mapping affect application performance for particular scenarios. Finally, we describe future research being performed at USC/ISI to develop an intelligent, dynamic run-time resource allocation strategy for large multi-core architectures.

System Model

We assume a multi-core architecture that has identical cores connected with a two dimensional mesh network. Each core has a processor, local memory, and a network switch. Store and forward packet switching is assumed to be used for the communication between cores. The network switch at each core is a 5x5 crossbar switch. An application may use multiple cores and is assumed to be able to change its configuration dynamically at run-time. The characteristics of an application may or may not be known ahead. Relevant characteristics include computational requirements, communication amounts, and a task graph (which specifies a communication topology). However, as an application changes its configuration at run-time, characteristics such as degree of parallelism, the computation/communication needs and communication pattern may also change dynamically. Such dynamicity makes resource allocation and performance tuning harder because an optimal allocation at one time will not continue to be optimal as an application changes its configuration dynamically.

This effort was sponsored by Defense Advanced Research Projects Agency (DARPA) through the Dept. of Interior National Business Center (NBC), under grant number NBCH1050022. This abstract is approThe U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsement, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Dept. of Interior, NBC, or the U.S. Government.

Simulation Environment

We built a high level performance simulator for large multi-core architectures using SystemC. Our simulator is based on the ARMn multiprocessor simulator [2], with major modifications. The network switch and network topology are adopted from ARMn. We built a generic processor module with a built-in proportional share resource scheduler. Clocked simulation is replaced with a high-level event driven simulation approach for faster simulation. Many system-dependent factors are run-time parameterized, including core CPU speed, network speed, CPU-network switch interface speed, packet size, overhead of control information in a packet, and network buffer size per input port of the network switch.

Simulation Scenario and Results

The application used in this study is derived from a Ground Moving Target Indicator (GMTI) radar processing stream.

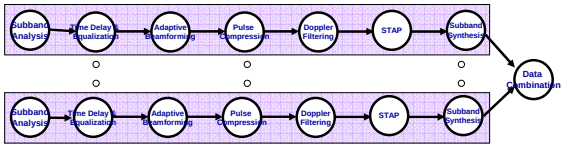


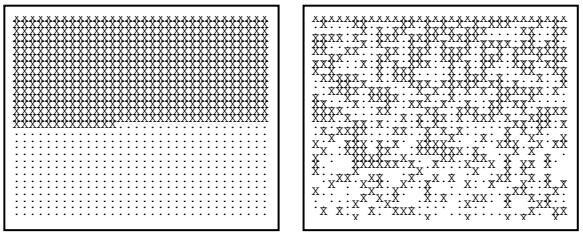
Figure 1. Task Graph of the Application Used

The flow graph is shown in Figure 1, where a circle denotes a task and arrows denote communication between tasks The task graph consists of a set of parallel chains with a sink node. The computation and the communication amounts are set such that each task consumes up to 7.5% of CPU time and a communication consumes up to 7.5% of the bandwidth of a network link. The application can change its computation/communication requirements by changing the number of the task chains, which effectively changes degree of parallelism.

Resource Fragmentation: To study the resource fragmentation, we used two applications. One application is GMTI. The other application is a model of a Feature-Aided Tracker (FAT) in which target density profiles in a spatial region are modeled as random processing loads over a set

of cores. The resource requirements of GMTI are varied by sequencing over a limited combination of modes. Modes are defined by sets of parameters. We limited mode changes to variations in one parameter so that the changes in computation requirements are gradual. The resource requirements of FAT are varied by randomly changing the total computational requirement of FAT within ±10% to simulate gradual changes in the number of targets in the field. GMTI is a higher priority task and cores are allocated to FAT only after the needs of the current GMTI mode are met. In each time step, we derived a new resource requirement and allocated or de-allocated cores as needed.

Our core allocation strategy used a simple linear resource allocation technique, which searches for and allocates the free cores from core (0, 0). The snapshots of the fragmented resources are shown in Figure 2(b), where 'X' denotes cores allocated to GMTI, '.' to FAT. Blank denotes free cores. As is shown in Figure 2, cores are highly fragmented within GMTI and FAT after 25 time steps.

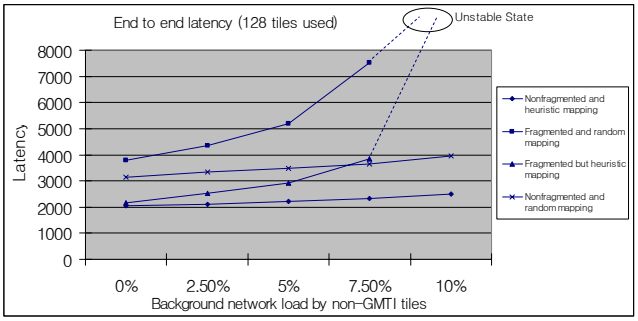


(a) Initial State (b) 25-th reconfiguration

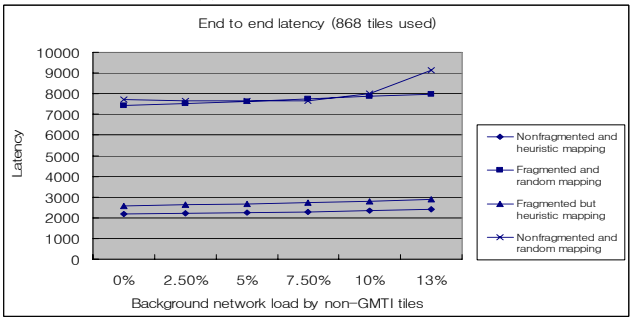
Figure 2. Resource Fragmentation

Application Mapping: Application mapping onto available resources affects the performance of an application. With the full knowledge of the characteristics of an application, including communication patterns, it is possible to do decent application mapping. However, without a priori knowledge of communication patterns, which is often the case in real world, an automated application mapping can be as bad as a random mapping. We performed experiments on four cases: 1) Heuristic mapping on clustered cores, 2) Random mapping on clustered cores, 3) Heuristic mapping on fragmented cores, 4) Random mapping on fragmented cores. We used the flow graph shown in Figure 1 for the applications mapped on to the system. Heuristic mapping maps the tasks in a chain onto nearby cores. The simulation was done for two sizes of the application, one occupying 128 of 1024 (32x32) cores and the other occupying 868 of 1024 (32x32) cores. For clustered experiments, 16x8 cores and 31x28 cores from the upper left corner are used. For fragmentation experiments, the cores are distributed randomly. End-to-end latency of the application is measured as a performance metric for comparison.

Simulation Results: We simulated network traffic interference from other by randomly generating network packets to and from non-GMTI cores. The network load generated by a non-GMTI core varies from 2.5% to 12.5% of the network link bandwidth. It should be noted that overall network load depends on the number of non-GMTI cores. For example, the experiment of 128 cores has about 7 times higher network load than that of 868 cores. For all cases, random mapping results in poor performance, which is expected. Mapping on fragmented resources tend to be more prone to network interference from other applications in the system. When the effect of network inference is small, application mapping dominates the performance, which is shown in Figures 3(a) and (b). Since the experiment of 868 cores uses most of the core for GMTI, it is less prone to the non-GMTI background network load. However, the quality of application mapping becomes more critical than the 128-core case.



(a) 128 cores are allocated



(b) 868 cores are allocated

Figure 3. End-to-end latency changes

Conclusion

In this research, we have studied resource fragmentation and application mapping for multi-core architectures in order to motivate the need for intelligent run-time systems to dynamically manage on-chip processing resources. We have also studied the manner in which inter-application and intra-application interferences through the shared network links can also degrade performance. Our hypothesis is that an intelligent run-time resource management technique can prevent resource fragmentation in dynamic applications and can improve performance over time by adjusting application mappings as necessary. The next phase of our research will include prototyping an intelligent run-time system. This system will introspect and monitor the behavior of applications on the multi-core architecture and use that information, combined with previous knowledge, to learn resource allocations and mappings which improve performance of applications over time.

References

[1] L. V. Kale, S. Kumar, and J. DeSouza, "A Malleable-Job System for Timeshared Parallel Machines," 2nd IEEE/ACM Int'l Symposium on Cluster Computing and the Grid, 2002.

[2] X.Zhu and S.Malik, "Using A Communication Architecture Specification in an Application-driven Retargetable Prototyping Platform for Distributed Processing", Proceedings of DATE 04, Feb, 2004.